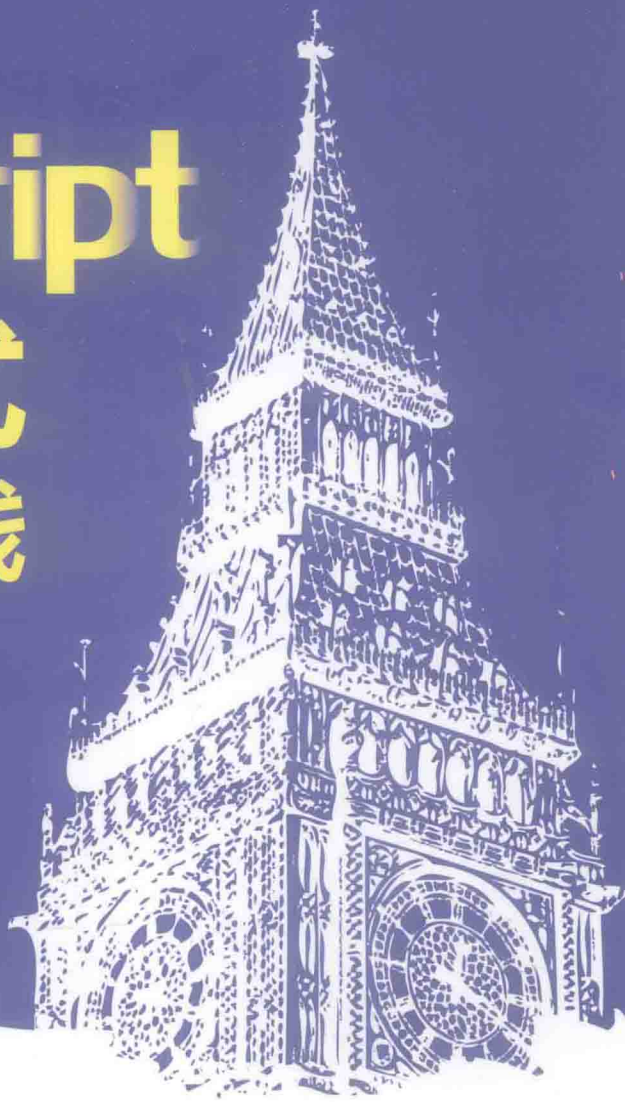
TURING 图灵原创

AlloyTeam出品

JavaScript 设计模式与开发实践



曾探◎著

全面涵盖专门针对JavaScript的16个设计模式

深入剖析面向对象设计原则、编程技巧及代码重构

中国工信出版集团

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵原创

JavaScript 设计模式与开发实践

曾探◎著

人民邮电出版社
北京

图书在版编目（CIP）数据

JavaScript设计模式与开发实践 / 曾探著. — 北京：人民邮电出版社，2015.5
（图灵原创）
ISBN 978-7-115-38888-9

Ⅰ. ①J… Ⅱ. ①曾… Ⅲ. ①JAVA语言－程序设计
Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2015)第072098号

内 容 提 要

本书根据 JavaScript 语言的特性，全面总结了实际工作中常用的设计模式。全书共分为三个部分，第一部分讲解了 JavaScript 语言面向对象和函数式编程的知识及其在设计模式方面的作用；第二部分通过一步步完善示例代码，由浅入深地讲解了 16 个设计模式；第三部分讲述了面向对象的设计原则及其在设计模式中的体现，以及一些常见的面向对象编程技巧和日常开发中的代码重构。

书中所有示例均来自作者长期的开发实践，与实际开发密切相关，适用于初、中、高级 Web 前端开发人员，尤其适合想往架构师晋级的中高级程序员阅读。

◆ 著　　曾　探
责任编辑　王军花
执行编辑　张　霞
责任印制　杨林杰
◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京天宇星印刷厂印刷
◆ 开本：800×1000　1/16
印张：19.5
字数：461千字　　2015年5月第1版
印数：1－4 000册　　2015年5月北京第1次印刷

定价：59.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京崇工商广字第 0021 号

序

如果时间倒退一点，很难想象我这样的“懒人”会花上近一年的业余时间来完成这本书。

这本书的原型是我发表在腾讯内部KM论坛的一篇文章《JavaScript常用设计模式》。这篇文章反响不错，还位列2012年KM十大热门文章第一名。不过说老实话，当时自己也是模式的初学者，和网上大部分讨论设计模式的文章一样，这篇文章里其实存在一些错误，这里要诚恳地说声抱歉。也正是由于这个原因，近两年我重新投身于对设计模式的研究之中。尽管如此，当在电脑上敲下本书第一行字的时候，我心中还是非常忐忑。一是我自己本身并非理论派，大部分工作时间都在做上层应用开发，很多偏理论的知识对于我来说，也是一个学习加总结的过程，二是不确保自己能否牺牲如此多的业余时间，毕竟很难削减玩LOL的时间。

无论如何，它终于和大家见面了。

本书结构

本书共分为三大部分。

第一部分讲解了JavaScript面向对象和函数式编程方面的知识，主要包括静态类型语言和动态类型语言的区别及其在实现设计模式时的异同，以及封装、继承、多态在动态类型语言中的体现，此外还介绍了JavaScript基于原型继承的面向对象系统的来龙去脉，给学习设计模式做好铺垫。

第二部分是核心部分，通过从普通到更好的代码示例，由浅到深地讲解了16个设计模式。

第三部分主要讲解面向对象的设计原则及其在设计模式中的体现，还介绍了一些常见的面向对象编程技巧和日常开发中的代码重构。

目标读者

本书主要面向初中级JavaScript开发人员。本书虽然以设计模式为主题，但也讲述了一些JavaScript开发中需要的基础知识，初级程序员也能从这里找到自己需要的东西。而对于中级程序员而言，学习设计模式的过程，可能正是往高级进阶的过程。

示例代码与勘误

本书提供了丰富的示例，示例代码可以在图灵社区本书主页（http://www.ituring.com.cn/book/1632）的“随书下载”中下载使用。

另外，由于作者的水平和时间所限，本书中难免存在一些遗憾。如果大家发现有什么问题，或者对本书有任何建议，欢迎到图灵社区本书主页提交勘误，也可以发送邮件到svenzeng@tencent.com来讨论，先谢谢！☺

致谢

虽然在写作过程中经历了不少曲折，但最终顺利完成。在这里，我想感谢为我提供帮助的所有人。

感谢图灵的美女编辑Alice，没有她的帮助，这本书不可能完成。

感谢AlloyTeam团队中每一个成员对我的指导和帮助，在这里工作不仅是工作，也是生活很重要的一部分。

感谢贺师俊、王集鹄、易郑超、程劭非几位老师提供的技术指导和宝贵建议。

感谢设计师“出壳设计”设计的插画和封面，它们让内容更加生动有趣。

最后，感谢我的妻子Annie，遇见你，是最美丽的意外。

前　言

《设计模式》一书自1995年成书一来，一直是程序员谈论的“高端”话题之一。许多程序员从设计模式中学到了设计软件的灵感，或者找到了问题的解决方案。在社区中，既有人对模式无比崇拜，也有人对模式充满误解。有些程序员把设计模式视为圣经，唯模式至上；有些人却认为设计模式只在C++或者Java中有用武之地，JavaScript这种动态语言根本就没有设计模式一说。

那么，在进入设计模式的学习之前，我们最好还是从模式的起源说起，分别听听这些不同的声音。

设计模式并非是软件开发的专业术语。实际上，“模式”最早诞生于建筑学。20世纪70年代，哈佛大学建筑学博士Christopher Alexander和他的研究团队花了约20年的时间，研究了为解决同一个问题而设计出的不同建筑结构，从中发现了那些高质量设计中的相似性，并且用“模式”来指代这种相似性。

受Christopher Alexander工作的启发，Erich Gamma、Richard Helm、Ralph Johnson、John Vlissides四人（人称Gang Of Four，GoF）把这种“模式”观点应用于面向对象的软件设计中，并且总结了23种常见的软件开发设计模式，录入《设计模式：可复用面向对象软件的基础》一书。

> 设计模式的定义是：在面向对象软件设计过程中针对特定问题的简洁而优雅的解决方案。

通俗一点说，设计模式是在某种场合下对某个问题的一种解决方案。如果再通俗一点说，设计模式就是给面向对象软件开发中的一些好的设计取个名字。

GoF成员之一John Vlissides在他的另一本关于设计模式的著作《设计模式沉思录》中写过这样一段话：

> 设想有一个电子爱好者，虽然他没有经过正规的培训，但是却日积月累地设计并制造出许多有用的电子设备：业余无线电、盖革计数器、报警器等。有一天这个爱好者决定重新回到学校去攻读电子学学位，来让自己的才能得到真实的认可。随着课程的展开，这个爱好者突然发现课程内容都似曾相识。似曾相识的并不是术语或者表述的方式，而是背后的概念。这个爱好者不断学到一些名称和原理，虽然这些名称和原理原来他不知道，但事实上他多年来一直都在使用。整个过程只不过是一个接一个的顿悟。

软件开发中的设计也是如此。这些“好的设计”并不是GoF发明的，而是早已存在于软件开发中。一个稍有经验的程序员也许在不知不觉中数次使用过这些设计模式。GoF最大的功绩是把这些“好的设计”从浩瀚的面向对象世界中挑选出来，并且给予它们一个好听又好记的名字。

那么，给模式一个名字有什么意义呢？上述故事中的电子爱好者在未进入学校之前，一点都不知道这些关于电器的概念有一些特定的名称，但这不妨碍他制造出一些电子设备。

实际上给“模式”取名的意义非常重要。人类可以走到生物链顶端的前两个原因分别是会“使用名字”和“使用工具”。在软件设计中，一个好的设计方案有了名字之后，才能被更好地传播，人们才有更多的机会去分享和学习它们。

也许这个小故事可以说明名字对于模式的重要性：假设你是一名足球教练，正在球场边指挥一场足球赛。通过一段时间的观察后，发现对方的后卫技术精湛，身体强壮，但边后卫速度较慢，中后卫身高和头球都非常一般。于是你在场边大声指挥队员：“用速度突破对方边后卫之后，往球门方向踢出高球，中路接应队员抢点头球攻门。”

在机会稍纵即逝的足球场上，教练这样费尽口舌地指挥队员比赛无疑是荒谬的。实际上这种战术有一个名字叫作“下底传中”。正因为战术有了对应的名字，在球场上教练可以很方便地和球员交流。“下底传中”这种战术即是足球场上的一种“模式”。

在软件设计中亦是如此。我们都知道设计经验非常重要。也许我们都有过这种感觉：这个问题发生的场景似曾相识，以前我遇到并解决过这个问题，但是我不知道怎么跟别人去描述它。我们非常希望给这个问题出现的场景和解决方案取一个统一的名字，当别人听到这个名字的时候，便知道我想表达什么。比如一个JavaScript新手今天学会了编写each函数，each函数用来迭代一个数组。他很难想到这个each函数其实就是迭代器模式。于是他向别人描述这个函数结构和意图的时候会遇到困难，而一旦大家对迭代器模式这个名字达成了共识，剩下的交流便是自然而然的事情。

学习模式的作用

小说家很少从头开始设计剧情，足球教练也很少从头开始发明战术，他们总是沿袭一些已经存在的模式。当足球教练看到对方边后卫速度慢，中后卫身高矮时，自然会想到“下底传中”这种模式。

同样，在软件设计中，模式是一些经过了大量实际项目验证的优秀解决方案。熟悉这些模式的程序员，对某些模式的理解也许形成了条件反射。当合适的场景出现时，他们可以很快地找到某种模式作为解决方案。

比如，当他们看到系统中存在一些大量的相似对象，这些对象给系统的内存带来了较大的负担。如果他们熟悉享元模式，那么第一时间就可以想到用享元模式来优化这个系统。再比如，系统中某个接口的结构已经不能符合目前的需求，但他们又不想去改动这个被灰尘遮住的老接口，一个熟悉模式的程序员将很快地找到适配器模式来解决这个问题。

如果我们还没有学习全部的模式，当遇到一个问题时，我们冥冥之中觉得这个问题出现的几率很高，说不定别人也遇到过同样的问题，并且已经把它整理成了模式，提供了一种通用的解决方案。这时候去翻翻《设计模式》这本书也许就会有意外的收获。

模式在不同语言之间的区别

《设计模式》一书的副标题是“可复用面向对象软件的基础”。《设计模式》这本书完全是从面向对象设计的角度出发的，通过对封装、继承、多态、组合等技术的反复使用，提炼出一些可重复使用的面向对象设计技巧。所以有一种说法是设计模式仅仅是就面向对象的语言而言的。

《设计模式》最初讲的确实是静态类型语言中的设计模式，原书大部分代码由C++写成，但设计模式实际上是解决某些问题的一种思想，与具体使用的语言无关。模式社区和语言一直都在发展，如今，除了主流的面向对象语言，函数式语言的发展也非常迅猛。在函数式或者其他编程范型的语言中，设计模式依然存在。

人类飞上天空需要借助飞机等工具，而鸟儿天生就有翅膀。在Dota游戏里，牛头人的人生目标是买一把跳刀（跳刀可以使用跳跃技能），而敌法师天生就有跳跃技能。因为语言的不同，一些设计模式在另外一些语言中的实现也许跟我们在《设计模式》一书中看到的大相径庭，这一点也不令人意外。

Google的研究总监Peter Norvig早在1996年一篇名为“动态语言设计模式”的演讲中，就指出了GoF所提出的23种设计模式，其中有16种在Lisp语言中已经是天然的实现。比如，Command模式在Java中需要一个命令类，一个接收者类，一个调用者类。Command模式把运算块封装在命令对象的方法内，成为该对象的行为，并把命令对象四处传递。但在Lisp或者JavaScript这些把函数当作一等对象的语言中，函数便能封装运算块，并且函数可以被当成对象一样四处传递，这样一来，命令模式在Lisp或者JavaScript中就成为了一种隐形的模式。

在Java这种静态编译型语言中，无法动态地给已存在的对象添加职责，所以一般通过包装类的方式来实现装饰者模式。但在JavaScript这种动态解释型语言中，给对象动态添加职责是再简单不过的事情。这就造成了JavaScript语言的装饰者模式不再关注于给对象动态添加职责，而是关注于给函数动态添加职责。

设计模式的适用性

设计模式被一些人认为只是夸夸其谈的东西，这些人认为设计模式并没有多大用途。毕竟我们用普通的方法就能解决的问题，使用设计模式可能会增加复杂度，或带来一些额外的代码。如果对一些设计模式使用不当，事情还可能变得更糟。

从某些角度来看，设计模式确实有可能带来代码量的增加，或许也会把系统的逻辑搞得更复杂。但软件开发的成本并非全部在开发阶段，设计模式的作用是让人们写出可复用和可维护性高

的程序。假设有一个空房间，我们要日复一日地往里面放一些东西。最简单的办法当然是把这些东西直接扔进去，但是时间久了，就会发现很难从这个房子里找到自己想要的东西，要调整某几样东西的位置也不容易。所以在房间里做一些柜子也许是个更好的选择，虽然柜子会增加我们的成本，但它可以在维护阶段为我们带来好处。使用这些柜子存放东西的规则，或许就是一种模式。

所有设计模式的实现都遵循一条原则，即“找出程序中变化的地方，并将变化封装起来”。一个程序的设计总是可以分为可变的部分和不变的部分。当我们找出可变的部分，并且把这些部分封装起来，那么剩下的就是不变和稳定的部分。这些不变和稳定的部分是非常容易复用的。这也是设计模式为什么描写的是可复用面向对象软件基础的原因。

设计模式被人误解的一个重要原因是人们对它的误用和滥用，比如将一些模式用在了错误的场景中，或者说在不该使用模式的地方刻意使用模式。特别是初学者在刚学会使用一个模式时，恨不得把所有的代码都用这个模式来实现。锤子理论在这里体现得很明显：当我们有了一把锤子，看什么都是钉子。拿足球比赛的例子来说，我们的目标只是进球，“下底传中”这种“模式”仅仅是达到进球目标的一种手段。当我们面临密集防守时，下底传中或许是一种好的选择；但如果我们的球员获得了一个直接面对对方守门员的单刀机会，那么是否还要把球先传向边路队友，再由边路队友来一个边路传中呢？答案是显而易见的，模式应该用在正确的地方。而哪些才算正确的地方，只有在我们深刻理解了模式的意图之后，再结合项目的实际场景才会知道。

分辨模式的关键是意图而不是结构

在设计模式的学习中，有人经常发出这样的疑问：代理模式和装饰者模式，策略模式和状态模式，策略模式和智能命令模式，这些模式的类图看起来几乎一模一样，它们到底有什么区别？

实际上这种情况是普遍存在的，许多模式的类图看起来都差不多，模式只有放在具体的环境下才有意义。比如我们的手机，把它当电话的时候，它就是电话；把它当闹钟的时候，它就是闹钟；用它玩游戏的时候，它就是游戏机。我看到有人手中拿着iPhone18，但那实际上可能只是一个吹风机。有很多模式的类图和结构确实很相似，但这不太重要，辨别模式的关键是这个模式出现的场景，以及为我们解决了什么问题。

对JavaScript设计模式的误解

虽然JavaScript是一门完全面向对象的语言，但在很长一段时间内，JavaScript在人们的印象中只是用来验证表单，或者完成一些简单动画特效的脚本语言。在JavaScript语言上运用设计模式难免显得小题大做。但目前JavaScript已成为最流行的语言之一，在许多大型Web项目中，JavaScript代码的数量已经非常多了。我们绝对有必要把一些优秀的设计模式借鉴到JavaScript这门语言中。许多优秀的JavaScript开源框架也运用了不少设计模式。

JavaScript设计模式的社区目前还几乎是一片荒漠。网络上有一些讨论JavaScript设计模式的

资料和文章，但这些资料和文章大多都存在两个问题。

第一个问题是习惯把静态类型语言的设计模式照搬到JavaScript中，比如有人为了模拟JavaScript版本的工厂方法（Factory Method）模式，而生硬地把创建对象的步骤延迟到子类中。实际上，在Java等静态类型语言中，让子类来“决定”创建何种对象的原因是为了让程序迎合依赖倒置原则（DIP）。在这些语言中创建对象时，先解开对象类型之间的耦合关系非常重要，这样才有机会在将来让对象表现出多态性。

而在JavaScript这种类型模糊的语言中，对象多态性是天生的，一个变量既可以指向一个类，又可以随时指向另外一个类。JavaScript不存在类型耦合的问题，自然也没有必要刻意去把对象“延迟”到子类创建，也就是说，JavaScript实际上是不需要工厂方法模式的。模式的存在首先是能为我们解决什么问题，这种牵强的模拟只会让人觉得设计模式既难懂又没什么用处。

另一个问题是习惯根据模式的名字去臆测该模式的一切。比如命令模式本意是把请求封装到对象中，利用命令模式可以解开请求发送者和请求接受者之间的耦合关系。但命令模式经常被人误解为只是一个名为execute的普通方法调用。这个方法除了叫作execute之外，其实并没有看出其他用处。所以许多人会误会命令模式的意图，以为它其实没什么用处，从而联想到其他设计模式也没有用处。

这些误解都影响了设计模式在JavaScript语言中的发展。

模式的发展

前面说过，模式的社区一直在发展。GoF在1995年提出了23种设计模式。但模式不仅仅局限于这23种。在近20年的时间里，也许有更多的模式已经被人发现并总结了出来。比如一些JavaScript图书中会提到模块模式、沙箱模式等。这些“模式”能否被世人公认并流传下来，还有待时间验证。不过某种解决方案要成为一种模式，还是有几个原则要遵守的。这几个原则即是“再现”“教学”和“能够以一个名字来描述这种模式”。

不管怎样，在一些模式被公认并流行起来之前，需要慎重地冠之以某种模式的名称。否则模式也许很容易泛滥，导致人人都在发明模式，这反而增加了交流的难度。说不准哪天我们就能听到有人说全局变量模式、加模式、减模式等。

在《设计模式》出版后的近20年里，也出现了另外一批讲述设计模式的优秀读物。其中许多都获得过Jolt大奖。数不清的程序员从设计模式中获益，也许是改善了自己编写的某个软件，也许是从设计模式的学习中更好地理解了面向对象编程思想。无论如何，相信对我们这些大多数的普通程序员来说，系统地学习设计模式并没有坏处，相反，你会在模式的学习过程中受益匪浅。

目　录

第一部分　基础知识

第二部分　设计模式